Program Assignment
LL(1) Parser

Compiler Theory #1

21600193

Hyo-Rim, Kim

Jun 23rd, 2019

1. BNF

program ⟶ stmt-sequence | ε

stmt-sequence ⟶ statement stmt-seq'

stmt-seq' ⟶ ; statement stmt-seq' | ε

statement ⟶ for-stmt | while-stmt | if-stmt | class-stmt | func-stmt | assign-stmt | main-stmt

main-stmt ⟶ main ( ) { stmt-sequence }

if-stmt ⟶ if (expr) { stmt-sequence } else-stmt

else-stmt ⟶ else else-stmt' | ε

else-stmt' ⟶ { stmt-sequence } | if-stmt else-stmt

class-stmt ⟶ class id { stmt-sequence }

func-stmt ⟶ func (func-parameter)

assign-stmt ⟶ id assign-stmt' | int id = expr

assign-stmt'⟶ = expr | ++ | --

for-stmt ⟶ for (assign-stmt ; expr ; assign-stmt) { stmt-sequence }

while-stmt ⟶ while (expr) { stmt-sequence }

expr ⟶ term expr'

expr' ⟶ addop term expr' | cmpop term expr' | ε

func-parameter ⟶ id func-parameter' | " literal literal' "

literal' ⟶ literal literal' | ε

func-parameter' ⟶ , id func-parameter' | ε
term ⟶ factor term'
term' ⟶ mulop factor term' | ε
addop ⟶ + | -
cmpop ⟶ < | <= | > | >= | == | !=
mulop ⟶ * | / | %
factor ⟶ number | id | (expr)

2. First Sets

program	ε, for, while, if, class, func, id, main, int
stmt-sequence	for, while, if, class, func, id, main, int
stmt-seq’	;, ε
statement	for, while, if, class, func, id, main, int
main-stmt	main
if-stmt	if
else-stmt	else, ε
else-stmt’	{, if
class-stmt	class
func-stmt	func
assign-stmt	id, int
assign-stmt’	=, ++, --
for-stmt	for
while-stmt	while
expr	number, id, (
expr'	ε, +, -, <, <=, >, >=, ==, !=
func-parameter	id, “
func-parameter’	,, ε
literal'	literal, ε
term	number, id, (
term'	ε, *, /, %
addop	+, -
cmpop	<, <=, >, >=, ==, !=
mulop	*. /, %
factor	number, id, (

3. Follow Sets

program	$
stmt-sequence	}, $
stmt-seq’	}, $
statement	;, }, $
main-stmt	;, }, $
if-stmt	else, ;, }, $
else-stmt	else, ;, }, $
else-stmt’	else, ;, }, $
class-stmt	;, }, $
func-stmt	;, }, $
assign-stmt	;,), }, $
assign-stmt’	;,), }, $
for-stmt	;, }, $
while-stmt	;, }, $
expr	), ;, }, $
expr'	), ;, }, $
func-parameter	)
func-parameter’	)
literal'	“
term	+, -, <, <=, >, >=, ==, !=,), ;, }, $
term'	+, -, <, <=, >, >=, ==, !=,), ;, }, $
addop	number, id, (
cmpop	number, id, (
mulop	number, id, (
factor	*, /, %, +, -, <, <=, >, >=, ==, !=,), ;, }, $

4. Parsing Table

1) Parsing Table with origin BNF

M[N, T]	while	if	for	else	func	main	class	int
program	program ⟶ stmt-sequence	program ⟶ stmt-sequence	program ⟶ stmt-sequence		program ⟶ stmt-sequence	program ⟶ stmt-sequence	program ⟶ stmt-sequence	program ⟶ stmt-sequence
stmt-sequence	stmt-sequence ⟶ statement stmt-seq'	stmt-sequence ⟶ statement stmt-seq'	stmt-sequence ⟶ statement stmt-seq'		stmt-sequence ⟶ statement stmt-seq'	stmt-sequence ⟶ statement stmt-seq'	stmt-sequence ⟶ statement stmt-seq'	stmt-sequence ⟶ statement stmt-seq'
stmt-seq’								
statement	statement ⟶ while-stmt	statement ⟶ if-stmt	statement ⟶ for-stmt		statement ⟶ func-stmt	statement ⟶ main-stmt	statement ⟶ class-stmt	statement ⟶ assign-stmt
main-stmt						main-stmt ⟶ main(){ stmt-sequence }		
if-stmt		if-stmt ⟶ if (expr) {stmt-sequence } else-stmt						
else-stmt				else-stmt ⟶ else else-stmt’ else-stmt ⟶ ε				
else-stmt'		else-stmt’ ⟶ if-stmt else-stmt						
class-stmt							class-stmt ⟶ class id {stmt-sequence }	class-stmt ⟶ class id {stmt-sequence }
func-stmt					func-stmt ⟶ func (func-			

					parameter)			
assign-stmt								assign-stmt ⟶ int id assign-stmt'
assign-stmt'								
for-stmt			for-stmt ⟶ for(assign-stmt ; expr ; assign-stmt) {stmt-sequence }					
while-stmt	while-stmt ⟶ while(expr){stmt-sequence }							
expr								
expr'								
func-parameter								
func-parameter'								
literal'								
term								
term'								
addop								
cmpop								
mulop								
factor								

id	number literal	addop	cmpop	mulop	assign	left cully brace
program ⟶ stmt-sequence						
stmt-sequence ⟶ statement stmt-seq'						

statement ⟶ assign-stmt						
						else-stmt' ⟶ { stmt-sequence }
assign-stmt ⟶ id assign-stmt'						
					assign-stmt'⟶ = expr	
expr ⟶ term expr'	expr ⟶ term expr'					
		expr' ⟶ addop term expr'	expr' ⟶ cmpop term expr'			
func-parameter ⟶ id func-parameter'						
term ⟶ factor term'	term ⟶ factor term'					
				term' ⟶ mulop factor term'		
		term' ⟶ ε	term' ⟶ ε			
		addop ⟶ + | -				
			cmpop ⟶ < | <= | > | >= | == | !=			
				mulop ⟶ * | /| %		
factor ⟶ id	factor ⟶ number					

right cully brace	left brace	right brace	semi-colon	quotation
stmt-seq' ⟶ ε				

			stmt-seq' ⟶ ; stmt-sequence	
else-stmt ⟶ ε			else-stmt ⟶ ε	
	expr ⟶ term expr’			
expr' ⟶ ε		expr' ⟶ ε	expr' ⟶ ε	
				func-parameter ⟶ " literal literal' "
		func-parameter' ⟶ ε		
				literal’ ⟶ ε
	term ⟶ factor term’			
term' ⟶ ε		term' ⟶ ε	term' ⟶ ε	
	factor ⟶ (			

literal	comma	assignop	$
			program ⟶ ε
			stmt-seq' ⟶ ε
			else-stmt ⟶ ε
		assign-stmt’⟶ ++ | --	
			expr’ ⟶ ε

	func-parameter' ⟶ , id func-parameter		
literal' ⟶ literal literal' literal' ⟶ ε			literal' ⟶ ε
			term' ⟶ ε

2) number mapping

- state

0	program
1	stmt-sequence
2	stmt-seq'
3	statement
4	main-stmt
5	if-stmt
6	else-stmt
7	else-stmt'
8	class-stmt
9	func-stmt
10	assign-stmt
11	assign-stmt'
12	for-stmt
13	while-stmt
14	expr
15	expr'
16	func-parameter
17	func-parameter'
18	literal'
19	term
20	term'
21	factor

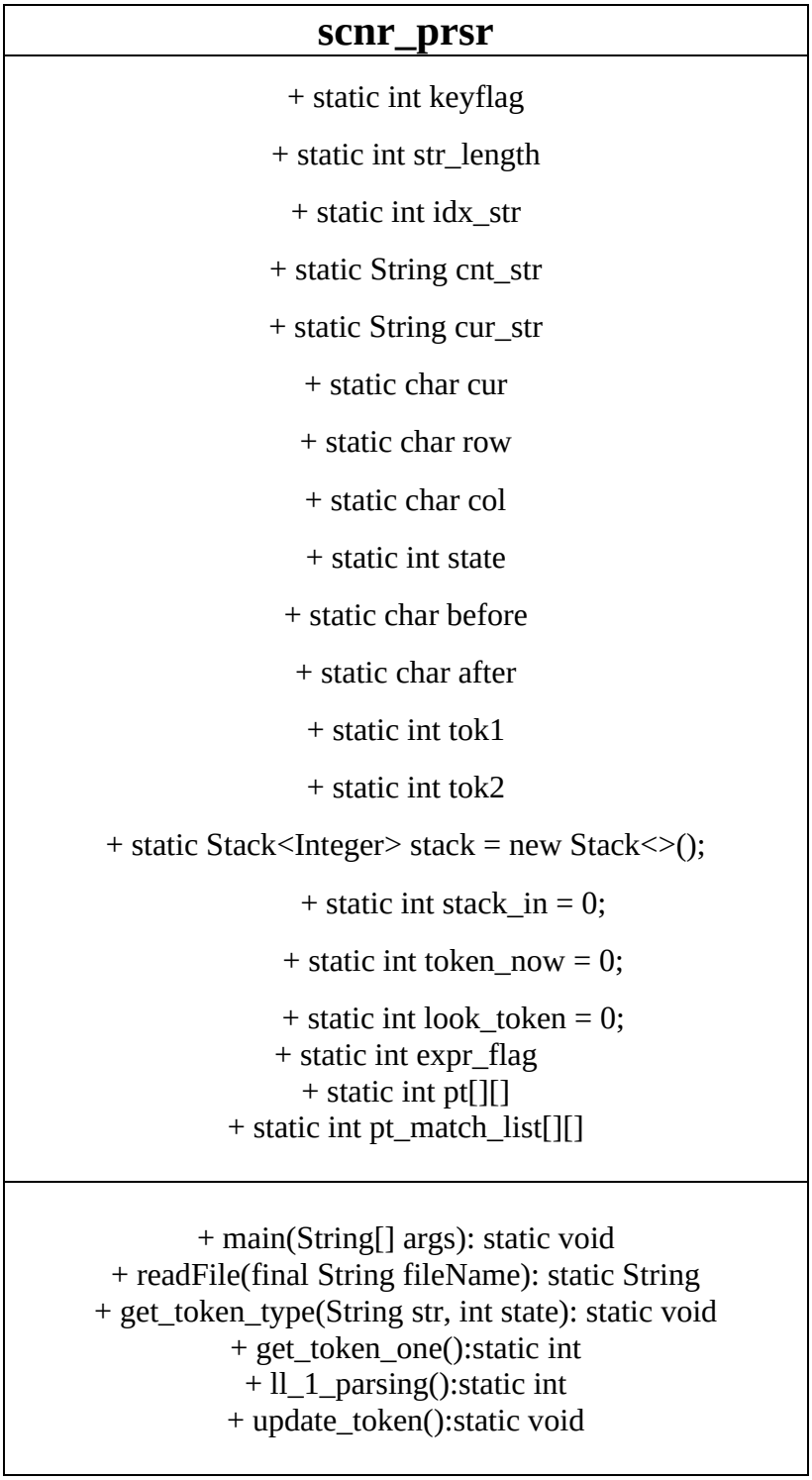
- input

0	while
1	if
2	for
3	else
4	func
5	main
6	class
7	int
8	id
9	number literal
10	addop
11	cmpop
12	mulop
13	assign
14	left cully brace
15	right cully brace
16	left brace
17	right brace
18	semi-colon
19	quotation
20	literal
21	comma
22	assignop

M[N, T]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	1	1	1	80	1	1	1	1	1	80	80	80	80	80	80	80	80	80	80	80	80	80	80	70
1	101	101	101	80	101	101	101	101	101	80	80	80	80	80	80	70	80	80	80	80	80	80	80	80
2	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	102	80	80	80	80	70
3	13	5	12	80	9	4	8	10	10	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
4	80	80	80	80	80	103	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
5	80	104	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
6	80	80	80	105	80	80	80	80	80	80	80	80	80	80	80	70	80	80	70	80	80	80	80	70
7	80	106	80	80	80	80	80	80	80	80	80	80	80	80	107	80	80	80	80	80	80	80	80	80
8	80	80	80	80	80	80	108	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
9	80	80	80	80	109	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
10	80	80	80	80	80	80	80	110	111	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
11	80	80	80	80	80	80	80	80	80	80	80	80	80	112	80	80	80	80	80	80	80	80	52	80
12	80	80	113	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
13	114	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
14	80	80	80	80	80	80	80	80	115	115	80	80	80	80	80	80	115	80	80	80	80	80	80	80
15	80	80	80	80	80	80	80	80	80	80	116	117	80	80	80	70	80	70	70	80	80	80	80	70
16	80	80	80	80	80	80	80	80	118	80	80	80	80	80	80	80	80	80	80	123	80	80	80	80
17	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	70	80	80	80	119	80	80
18	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	70	120	80	80	70
19	80	80	80	80	80	80	80	80	121	121	80	80	80	80	80	80	121	80	80	80	80	80	80	80
20	80	80	80	80	80	80	80	80	80	80	70	70	122	80	80	70	80	70	70	80	80	80	80	70
21	80	80	80	80	80	80	80	80	38	39	80	80	80	80	80	80	46	80	80	80	80	80	80	80

5. Design Document

1. UML



- get_token_type: get token with type
- ll_1_parsing: do LL(1) Parsing with pt[][](parsing table) and pt_match_list[][](input match list)
- update_token: get current and lookahead tokens

6. Result

1. Use Java

```
hyorm@ariselab:~/cmpr/hw3_21600193/use_javac/src$ java scnr_prsr ../data/test.txt
class : keyword
MyClass : id
{ : left curly brace
main : keyword
( : left parenthesis
) : right parenthesis
{ : left curly brace
int : keyword
$_Time0 : id
= : assignment symbol
22 : number literal
; : semicolon
if : keyword
( : left parenthesis
$_Time0 : id
< : greater than symbol
10 : number literal
) : right parenthesis
{ : left curly brace
out.println : keyword
( : left parenthesis
" : double quote symbol
Good : literal
morning. : literal
" : double quote symbol
) : right parenthesis
; : semicolon
} : right curly brace
else : keyword
if : keyword
( : left parenthesis
$_Time0 : id
< : greater than symbol
20 : number literal
) : right parenthesis
{ : left curly brace
out.println : keyword
( : left parenthesis
" : double quote symbol
Good : literal
day. : literal
" : double quote symbol
) : right parenthesis
; : semicolon
} : right curly brace
else : keyword
{ : left curly brace
out.println : keyword
( : left parenthesis
" : double quote symbol
Good : literal
evening. : literal
" : double quote symbol
) : right parenthesis
; : semicolon
} : right curly brace
} : right curly brace
} : right curly brace
Parsing Ok
```

2. Use Ant

```
init:

build:
    [javac] Compiling 1 source file to /home/hyorm/cmpr/hw3_21600193/use_ant/build

run:
     [java] class : keyword
     [java] MyClass : id
     [java] { : left curly brace
     [java] main : keyword
     [java] ( : left parenthesis
     [java] ) : right parenthesis
     [java] { : left curly brace
     [java] int : keyword
     [java] $_Time0 : id
     [java] = : assignment symbol
     [java] 22 : number literal
     [java] ; : semicolon
     [java] if : keyword
     [java] ( : left parenthesis
     [java] $_Time0 : id
     [java] < : greater than symbol
     [java] 10 : number literal
     [java] ) : right parenthesis
     [java] { : left curly brace
     [java] out.println : keyword
     [java] ( : left parenthesis
     [java] " : double quote symbol
     [java] Good : literal
     [java] morning. : literal
     [java] " : double quote symbol
     [java] ) : right parenthesis
     [java] ; : semicolon
     [java] } : right curly brace
     [java] else : keyword
     [java] if : keyword
     [java] ( : left parenthesis
     [java] $_Time0 : id
     [java] < : greater than symbol
     [java] 20 : number literal
     [java] ) : right parenthesis
     [java] { : left curly brace
     [java] out.println : keyword
     [java] ( : left parenthesis
     [java] " : double quote symbol
     [java] Good : literal
     [java] day. : literal
     [java] " : double quote symbol
     [java] ) : right parenthesis
     [java] ; : semicolon
     [java] } : right curly brace
     [java] else : keyword
     [java] { : left curly brace
     [java] out.println : keyword
     [java] ( : left parenthesis
     [java] " : double quote symbol
     [java] Good : literal
     [java] evening. : literal
     [java] " : double quote symbol
     [java] ) : right parenthesis
     [java] ; : semicolon
     [java] } : right curly brace
     [java] } : right curly brace
     [java] } : right curly brace
     [java] Parsing Ok

BUILD SUCCESSFUL
Total time: 2 seconds
```

7. User Manual
    1. Use ant (directory name)
        1. ant version
            - Apache Ant(TM) version 1.10.5 compiled on March 28 2019
    2. build.xml

```
<project name="scnr_prsr" default="build" basedir=".">

    <property name="src" value="src"/>
    <property name="build" value="build"/>
    <property name="doc" value="doc"/>

    <path id="lib.path">
        <pathelement location="${build}" />
    </path>

    <target name="init">
        <mkdir dir="${build}"/>
    </target>
    <target name="build" depends="init">
        <javac srcdir="${src}" destdir="${build}" debug="true"
includeantruntime="false">
        </javac>
    </target>

    <target name="run" depends="build">
        <java classname="scnr_prsr" fork="true" dir="." maxmemory="4096m">
            <classpath location="."/>
            <classpath refid="lib.path"/>
            <arg file="data/test.txt"/>
        </java>
    </target>

    <target name="clean">
        <delete dir="${build}"/>
```

```
        </target>
</project>
```

8. command
    - ant build
    - ant run
        - this build.xml already set the file name(test.txt)
9. Use Javac (directory name)
    1. java version
        - openjdk version "11.0.6" 2020-01-14
        - OpenJDK Runtime Environment (build 11.0.6+10-post-Ubuntu-1ubuntu118.04.1)
        - OpenJDK 64-Bit Server VM (build 11.0.6+10-post-Ubuntu-1ubuntu118.04.1, mixed mode)
    2. command
        - javac scnr.java
        - java scnr [file name]